Chapter 11
Vision-Based Quadcopter Navigation in Structured Environments

Előd Páll, Levente Tamás, and Lucian Buşoniu

Abstract Quadcopters are small-sized aerial vehicles with four fixed-pitch propellers. These robots have great potential since they are inexpensive with affordable hardware, and with appropriate software solutions they can accomplish assignments autonomously. They could perform daily tasks in the future, such as package deliveries, inspections, and rescue missions. In this chapter, after an extensive introduction to object recognition and tracking, we present an approach for vision-based autonomous flying of an unmanned quadcopter in various structured environments, such as hallway-like scenes. The desired flight direction is obtained visually, based on perspective clues, in particular the vanishing point. This point is the intersection of parallel lines viewed in perspective, and is sought on the front camera image. For a stable guidance the position of the vanishing point is filtered with different types of probabilistic filters, such as linear Kalman filter, extended Kalman filter, unscented Kalman filter and particle filter. These are compared in terms of the tracking error and also for computational time. A switching control method is implemented. Each of the modes focuses on controlling only one state variable at a time and the objective is to center the vanishing point on the image. The selected filtering and control methods are tested successfully, both in simulation and in real indoor and outdoor environments.

11.1 Introduction

Unmanned aerial vehicles (*UAV*) are being increasingly investigated not only for military uses, but also for civilian applications. Famous recent examples are UAV applications for package delivery services, including those by online giant Amazon[1]

The authors are with the Department of Automation, Technical University of Cluj-Napoca, Memorandumului 28, 400114 Cluj-Napoca,Romania, e-mail: Pall.Elod@gmail.com, Levente.Tamas@aut.utcluj.ro, Lucian.Busoniu@aut.utcluj.ro

[1] Amazon testing drones for deliveries, BBC News 2013

and in the United Arab Emirates (Pavlik, 2014), or Deutsche Bahn's exploration of UAVs to control graffiti spraying[2]. More classical applications have long been considered, such as search and rescue (Beard et al, 2013; Tomic et al, 2012) or monitoring and fertilizer spraying in agriculture (Odido and Madara, 2013). Motivated by this recent interest, we focus here on the automation of a low-cost *quadcopters* such that they can perform tasks in structured environments without human interaction. We employ the *AR.Drone*, a lightweight UAV widely used in robotic research and education (Krajník et al, 2011; Stephane et al, 2012; Benini et al, 2013).

In particular, the aim of this chapter is to give an approach and implementation example where a UAV flies through corridor-like environments without obstacles, while using only on-board sensors. The UAV will not need a map of the surroundings or any additional markers to perform its task.

We use the forward looking camera and the Inertial Measurement Unit (IMU) to sense the surroundings. The application is based on *perspective vision* clues, namely on *vanishing point* detection. This point appears on a 2D projection of a 3D plane, where parallel lines viewed in perspective are intersecting each other. For better perception we are going to use Gaussian and non-parametric filters to track the position of the vanishing point. Another improvement of the sensing is achieved with *sensor fusion*, which is implemented directly in the filtering techniques by combining the rotational angle measurements with the detected visual information.

The estimated position of the vanishing point is going to be the input for our controller. We present a switching control strategy, which stabilizes the UAV in the middle of the hallway, while the UAV is flying toward the end of the corridor. The controller switches between the regulation of the yaw angle and of the lateral position of the quadcopter.

The fields of control (Hua et al, 2013) and vision-based state estimation (Shen et al, 2013) for UAVs are very well developed, as is robotic *navigation* (Bonin-Font et al, 2008; Bills et al, 2011; Majdik et al, 2013). For instance, close to our approach are the lane marker detection method in Ali (2008), the vanishing point-based road following method of Liou and Jain (1987), and the vision-based object detection on railway tracks in Rubinsztejn (2011).

Our method is based on Lange et al (2012) and we add advanced filtering methods such as linear and nonlinear *Kalman filters* to the vision-based detection. We present existing simplified models which describe the dynamics of a 2D point on an image. These models are used for position tracking. We generalize the solution of autonomous flight to both indoor and in hallway-like outdoor environments. The image processing steps are the same with slight modifications in the parameters setup, while the filtering and the control remains unchanged for both environments.

Next, we present the quadcopter used in Section 11.2. Then, we provide detailed methodological and theoretical background in Section 11.4. The approach and the implementation details are given in Section 11.5. We also present experimental results in indoor and outdoor corridor-like environments in Section 11.6. Finally, we conclude the chapter in Section 11.7.

[2] German railways to test anti-graffiti drones. BBC News 2013

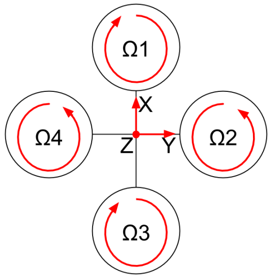


Fig. 11.1: Simple quadcopter schematics, where Ω_1, Ω_2, Ω_3, and Ω_4 are the propellers rotational speed.

11.2 Quadcopter Structure and Control

Quadcopters are classified as rotorcraft aerial vehicles. These mobile robots are frequently used in research because of their high maneuverability and agility.

The quadcopter‘s frame can be shaped as an **x** or **+** and the latter is presented in Figure 11.1. A brushless direct current motor is placed at each end of the configuration. The motors are rotating the fixed pitch propellers through a gear reduction system in order to generate lift. If all the motors are spinning with the same speed, $\Omega_1 = \Omega_2 = \Omega_3 = \Omega_4$, and the lift up force is equal with the weight of the quadcopter, then the drone is hovering in the air. This motor speed is called hovering speed Ω_h. Four basic movements can be defined in order to control the flight of the quadcopter:

- Vertical translational movement (U_1) is obtained by setting:

$$\Omega_1 = \Omega_2 = \Omega_3 = \Omega_4 \begin{cases} < \Omega_h & \text{losing altitude} \\ > \Omega_h & \text{gaining altitude} \end{cases}$$

- Roll movement is the rotation around **X** axis (U_2): $\begin{cases} \Omega_2 \neq \Omega_4 \\ \Omega_1 = \Omega_3 \end{cases}$
- Pitch movement is the rotation around **Y** axis (U_3): $\begin{cases} \Omega_2 = \Omega_4 \\ \Omega_1 \neq \Omega_3 \end{cases}$
- Yaw movement is the rotation around **Z** axis (U_4): $\Omega_1 = \Omega_2 \neq \Omega_3 = \Omega_4$

The horizontal translation of the quadcopter is achieved by pitching for forward and backward flight and by rolling for lateral translations. When the drone is tilted on an axis, it flies in the direction of the tilt.

From the control point of view, the quadcopter is controlled with the above presented four kind of inputs (U_1, U_2, U_3, U_4). The first three are translational movement commands, while the last one, U_4 is the rotational control input. Moreover the output of the quadcopter system is the 3D coordinates and orientation angles of the drone.

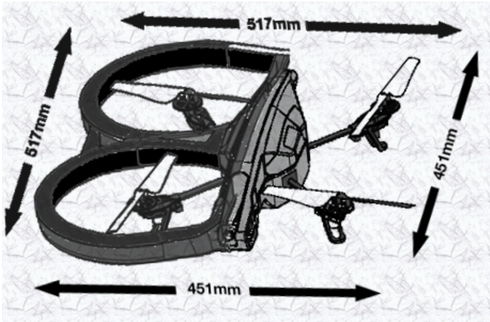


Fig. 11.2: Parrot AR.Drone schematics with and without indoor hull[3].

Regarding the mathematical dynamic model, the quadcopter has six degrees of freedom and the model has twelve parameters (linear and angular position and velocity) to describe the state of the vehicle. The full model is based on the Newton-Euler model as presented in (Bresciani, 2008). This is a complex model with a hybrid reference frame, where the translational motion equations are expressed with respect to the world frame and the rotational motion equations are given with respect to the quadcopter body frame. Instead of this, we are going to use a simple model, presented in Section 11.4.2, which will include only the yaw rotation of the quadcopter combined with a constant velocity model, because we are not focusing on quadcopter flight stabilization control, but on higher level control.

11.3 Quadcopter Hardware and Software

The chosen quadcopter is the Parrot AR.Drone 2.0, see Figure 11.2 for a schematic. It is a low-cost but well-equipped drone suitable for fast development of research applications (Krajník et al, 2011). The drone has a 1 GHz 32 bit ARM Cortex A8 processor with dedicated video processor at 800 MHz. A Linux operating system is running on the on board micro-controller. The AR.Drone is equipped with an IMU composed of a 3 axis gyroscope with 2000 °/second precision, a 3 axis accelerometer with ± 50 mg precision, and a 3 axis magnetometer with 6° precision. Moreover, it is supplied with two cameras, one at the front of the quadcopter having a wide angle lens, 92° diagonal and streaming 720p signal with 30 fps. The second camera is placed on the bottom, facing down. For lower altitude measurements an ultrasound sensor is used, and for greater altitude a pressure sensor with ± 10 Pa precision is employed.

Parrot designed the AR.Drone also for hobby and smart-phone users. Therefore, the drone has a WiFi module for communication with mobile devices. The stabiliza-

[3] image based on: http://ardrone2.parrot.com/ardrone-2/specifications/

tion and simple flight control (roll, pitch, yaw, and altitude) is already implemented on the micro-controller and it is not recommended to make changes in the supplied software. The quadcopter can stream one camera feed at a time together with all the rest of sensor data.

For research purposes an off-board control unit should be used because of the limitations of the on-board controller. The Robotic Operating System (ROS) (Quigley et al, 2009) is suitable to create control applications for mobile robots, moreover it has libraries to communicate with the drone. This is a meta-operating system with a Unix-like ecosystem. For our purpose is suitable because supports distributed systems, it has low level device abstraction, and includes an AR.Drone driver package to control and read sensor data .

Simulation is a powerful tool in the testing phase of a project, when image processing and control algorithms can be validated. We choose Gazebo, it is a simulation tool, compatible with ROS, and it has the dynamic model of the AR.Drone. This tool simulates the dynamics and he sensors of the drone, while ground truth information is also available.

11.4 Methodological and Theoretical Background

In this section, we present the methods taken from the literature that we employ in this research project, with their theoretical background. Our goal is to fly the drone autonomously in corridor-like environments by using only the sensors on the AR.Drone.

In the *mobile robotics* field, the automation of vehicles needs to solve the localization problem. In order to decide which type of localization should be used, the available sensors must be known. The AR.Drone is not necessarily supplied with global positioning sensors (GPS) and even if it is, the GPS can not be used indoor. Hence, we chose vision-based position tracking based on feature detection, which is presented in Section 11.4.1. This method will supply information about the relative position of the drone and the target.

In particular, the desired flight direction is represented by the vanishing point, a 2D point on the image. The position of the detected point is going to be tracked, because it changes in a dynamic fashion due to the drone motion, and it is affected by noise on the images. The chosen tracking methods are probabilistic state estimator filters, detailed in Section 11.4.2. These methods support sensor fusion, meaning that information from different sensors is combined. In our case, we fuse visual information with IMU measurements, using to enhance the precision of the estimation.

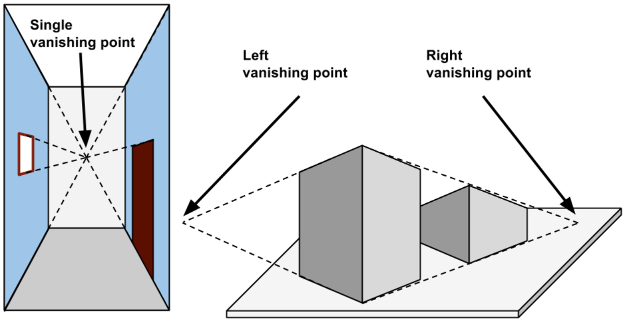


Fig. 11.3: Single VP (left) detected in a corridor and multiple VPs (right) detected when buildings are present.

11.4.1 Feature Detection

The AR.Drone has two color cameras, one at the bottom facing down and the other at the front, looking ahead. For the experiments we use the front camera.

The localization procedure should be general in the sense that no artificial changes, such as landmarks should be made to the environment in order to track the position of the quadcopter. Hence, the perspective view's characteristics are analyzed on the 2D image. One of the basic features of a 2D projection of 3D space is the appearance of vanishing points (VPs). The interior or the outdoor view of a building has many parallel lines and most of these lines are edges of buildings. Parallel lines projected on an image can converge in perspective view to a single point, called VP, see Figure 11.3. The flight direction of the quadcopter is determined by this 2D point on the image.

In order to obtain the pixel coordinates of the VP, first the image is pre-processed, next the edges of the objects are drawn out, and finally the lines are extracted based on the edges.

Pre-processing

In order to process any image, we must first calibrate the camera. Distortions can appear on the image due to the imperfections of the camera and its lens. Our project is based on gray-scale images processing. Therefore we are not focusing on the color calibration of the camera, and only spatial distortions are handled. The calibration parameters are applied on each frame. The calibration process has two parts: intrinsic and extrinsic camera parameters.

The general equation which describes the perspective projection of a 3D point on a 2D plane is:

Fig. 11.4: Camera calibration, on the left the barrel distorted image, on the right the calibrated image.

$$s \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \text{; and } \mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where $(u,\ v)$ are the projected point coordinates, s is the skew factor, and $\mathbf{K}$ is a camera matrix with the intrinsic parameters, focal length f_x, f_y and the optical centers c_x, c_y. These five parameters must be defined for a specific resolution of the camera. $[\mathbf{R}|\mathbf{t}]$ is a joint rotational-translational matrix with the extrinsic parameters, and $(X,\ Y,\ Z)$ are the 3D coordinates of the point in the world frame.

The spatial distortion is corrected with the extrinsic parameters. Barrel distortion (Zhang, 2000) appears on the front camera, because the camera is equipped with a wide angle lens. The distortion appears when an image is mapped around a sphere, as shown on Figure 11.4. The straight lines are curved on the distorted image (left) which is unacceptable for line-detection-based position tracking. The barrel effect has a radial distortion that can be corrected with:

$$\begin{aligned} x_c^b &= x_u(1+k_1r^2+k_2r^4+k_3r^6) \\ y_c^b &= y_u(1+k_1r^2+k_2r^4+k_3r^6) \end{aligned} \tag{11.1}$$

where (x_u, y_u) are the coordinates of a pixel point on the distorted image, $r = x_u^2 + y_u^2$, and (x_c^b, y_c^b) is the radial corrected position.

The tangential distortion appears when the lens is not perfectly parallel to the sensor in the digital camera. The correction of this error is expressed as:

$$\begin{aligned} x_c^t &= x_c^b + [2p_1x_uy_u + p_2(r^2+2x_u^2)] \\ y_c^t &= y_c^b + [p_1(r^2+2y_u^2) + 2p_2x_uy_u] \end{aligned} \tag{11.2}$$

where (x_c^t, y_c^t) is the tangential corrected position.

The extrinsic distortion coefficients appear in (11.1) and (11.2), where k_n is the n^{th} radial distortion coefficient and p_n is the n^{th} tangential distortion coefficient.

After obtaining the calibrated image some low-level intermediate pre-processing can be applied. In particular, we will use:

- smoothing, which eliminates noise and small fluctuations on images, it is equivalent with a low-pass filter in the frequency domain, and its drawback is blurring the sharp edges;
- gradient operators, which sharpen the image and act like a high-pass filter in the frequency domain.

Edge detection is based on gradient operators and it is considered to be a pre-processing step. An edge on a 2D image can bee seen as a strong change of intensity between the two surfaces i.e. as a first derivative maximum or minimum. For easier detection, a gradient filter is applied on a gray-scale image. As this is a high-pass filters, gradient operator also increases the noise level on the image.

We choose the Canny algorithm, it is a multi-step edge detector. First, it smooths the image with a Gaussian filter, and then finds the intensity gradient of the image by using the Sobel method. The Sobel algorithm is a first order edge detector, and it performs a 2D spatial gradient on the image.

Line detection

Sharp and long edges can be considered as lines on a gray-scale image. The most frequently used line detector is the Hough Transformation (HT) (Kiryati et al, 1991). This algorithm performs a grouping of edge points to obtain long lines from the output of the edge detector algorithm. The preliminary edge detection may be imperfect such as presenting missing points or spatial deviation from the ideal line. The HT method is based on the parametric representation of a line: $\rho = x\cos\theta + y\sin\theta$ where ρ is the perpendicular distance from the origin to the line and θ is the angle between the horizontal axis and the perpendicular line to the line to be detected. Both variables are normalized such that $\rho \geq 0$ and $\theta \in [0,\ 2\pi)$.

The family of lines, $L_{x_0 y_0}$, going through a given point (x_0, y_0) can be written as a set of pairs of $(\rho_\theta,\ \theta)$. Based on our previous normalization, $L_{x_0 y_0}$ can be represented as a sinusoid for the function $f_\theta(\theta) = \rho_\theta$. If two sinusoidal curves for $L_{x_a y_a}$ and $L_{x_b y_b}$ are intersecting each other at a point $(\rho_{\theta_{ab}},\ \theta_{ab})$, then the two points $(x_a\ y_a)$ and $(x_b\ y_b)$ are on the line defined with the parameters, $\rho_{\theta_{ab}}$ and θ_{ab}. The algorithm searches for intersections of sinusoidal curves. If the number of curves in the intersection is more than a given threshold, then the pair (ρ_θ, θ) is considered to be a line on the image.

The Probabilistic Hough Transform (PHT) is an improvement of HT (Stephens, 1991). The algorithm takes a random subset of points for line detection, therefore the detection is performed faster.

11.4.2 Feature Tracking

The true position of a moving object cannot be based only on observation, because measurement errors can appear even with the most advanced sensors. Hence, the position should be estimated based not only on the measurements, but also taking in consideration other factors such as the characteristics of the sensor, its performance and the dynamics of the moving object. Position tracking of a moving object with a camera is a well studied problem (Peterfreund, 1999; Schulz et al, 2001). In our case the estimators are used to reduce the noise in the VP detection and to track the position of the VP with higher confidence.

The estimation process can be enhanced by sensor fusion, meaning that different types of data are combined in order to have a more precise estimation of the current state. The 2D coordinates of the VP on the video feed is one measurement data and the other is the yaw orientation angle of the drone w.r.t. the body frame obtained from the IMU.

Next we present the models used to approximate the horizontal movement of the VP. Then we give a short description of the employed linear and nonlinear estimation methods, namely the linear Kalman filter, extended Kalman filter, unscented Kalman filter, and *particle filter*.

Vanishing point motion models

Generally, the state-space model of a system with additive Gaussian noise is defined as:

$$\begin{aligned} \mathbf{x}_{k+1} &= f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k, \ \mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k) \\ \mathbf{z}_k &= h(\mathbf{x}_k) + \mathbf{v}_k, \qquad \mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k) \end{aligned} \tag{11.3}$$

where $\mathbf{x}$ is the state variable, k is the step, $\mathbf{u}$ is the command signal, $\mathbf{z}$ is the measured output of the system, f is the nonlinear system function, $\mathbf{w}$ is the model noise, and $\mathbf{v}$ is the measurement noise. The model and measurement noises $\mathbf{Q}$, $\mathbf{R}$ are assumed to have a Gaussian distribution and to be uncorrelated.

We are going to model the dynamics of the detected VP. While, the VP is fixed in the earth frame, it moves w.r.t. the body frame. The dynamics of the camera are known, based on the motion model of the quadcopter. The camera is translated on the X axis in the body frame. Despite the fact that the quadcopter dynamic model is known, we are going to use a simplified VP motion model. For the following reason the altitude of the quadcopter is going to be kept constant, moreover the roll and pitch angular velocities have a small variation. Hence the vertical position of the VP can be neglected, and only the horizontal position should be tracked. Our simplified model is not going to use the homography mapping between the 3D quadcopter motion model and the 2D VP motion on the image, in order to reduce the complexity of the calculations.

Two simplified motion models were adopted for horizontal movement. The first is called the constant velocity model and it is a simple linear model (Durrant-Whyte, 2001). The state variable is composed of the one dimensional coordinate and velocity on the y axis, $\mathbf{x}_k = (y_k, \dot{y}_k)$, recall Figure 11.1. The model is a random walk model for the velocity, meaning we do not assume a dynamics for the velocity and just rely on the measurements:

$$\begin{cases} y_{k+1} = y_k + \dot{y}_k \cdot \delta_k + w_k^y \\ \dot{y}_{k+1} = \dot{y}_k + w_k^{\dot{y}} \end{cases} \tag{11.4}$$

with δ_k being the sampling time at step k. The sampling time may vary due to communication latency between the quadcopter and the ground station. The process and measurement covariance matrices are defined as follows:

$$\mathbf{Q} = \begin{bmatrix} \delta_k^4/3 & \delta_k^3/2 \\ \delta_k^3/2 & \delta_k \end{bmatrix} \cdot \sigma_w^2;\ R = \sigma_v^2$$

where σ_w and σ_v are standard deviation tuning parameters for the process and the measurement noise (Durrant-Whyte, 2001).

The second simplified model extends the constant velocity model with the yaw rotational angle, ϕ, making the model nonlinear. This model implements the fusion between the angular position measured with the IMU sensors and the VP detected on the video feed from the front camera. The state is $\mathbf{x}_k = (y_k,\ \dot{y}_k,\ \phi_k)$ and the model is:

$$\begin{cases} y_{k+1} &= y_k + sin(\phi_k) \cdot \dot{y}_k \cdot \delta_k + w_k^y \\ \dot{y}_{k+1} &= \dot{y}_k + w_k^{\dot{y}} \\ \phi_{k+1} &= \phi_k + w_k^{\phi} \end{cases} \tag{11.5}$$

Moreover, the process and measurement covariance matrices are defined as (Durrant-Whyte, 2001):

$$\mathbf{Q} = \begin{bmatrix} \delta_k^5/20 & \delta_k^4/8 & \delta_k^3/6 \\ \delta_k^4/8 & \delta_k^3/3 & \delta_k^2/2 \\ \delta_k^3/6 & \delta_k^2/2 & \delta_k \end{bmatrix} \cdot \sigma_w^2;\ R = \sigma_v^2$$

The third model is an extension of the constant velocity model with acceleration, so it is called constant acceleration model and described as follows:

$$\begin{cases} y_{k+1} &= y_k + \dot{y}_k \cdot \delta_k + \ddot{y}_k \cdot \frac{\delta_k^2}{2} + w_k^y \\ \dot{y}_{k+1} &= \dot{y}_k + \ddot{y}_k \cdot \delta_k + w_k^{\dot{y}} \\ \ddot{y}_{k+1} &= \ddot{y}_k + w_k^{\ddot{y}} \end{cases} \tag{11.6}$$

The process and measurement covariance matrices are defined as (Durrant-Whyte, 2001):

$$\mathbf{Q} = \begin{bmatrix} \delta_k^4/3 & \delta_k^3/2 & 0 \\ 0 & \delta_k^3/2 & \delta_k \\ 0 & 0 & 1 \end{bmatrix} \cdot \sigma_w^2;\ R = \sigma_v^2 \tag{11.7}$$

These models will be used in the Kalman filters to predict and estimate the position of the VP.

Linear Kalman filter

The linear Kalman filter (LKF) also known as linear quadratic estimation (LQE) (Durrant-Whyte, 2001) and it was designed for linear discrete time systems.

In case (11.3) is linear, we can represent it like:

$$\begin{aligned} \mathbf{x}_k &= \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \\ \mathbf{z}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{v}_k \end{aligned} \tag{11.8}$$

where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are the system matrices, $\mathbf{z}$ is the measurement, while $\mathbf{w}$ and $\mathbf{v}$ are the process and measurement noises with covariances $\mathbf{Q}$ and $\mathbf{R}$. The LKF is a suitable choice for object tracking problems because it is an optimal estimator for model 11.8 and moreover it can be used for sensor fusion.

The algorithm estimates the state $\mathbf{x}_k$ recursively and it has two steps. First, a prediction of the current state is calculated with the help of the system's model. In the second step the estimate is calculated by updating the prediction with the measurement, using the recalculated probability distribution.

Before using the LKF, an initial state, $\mathbf{x}_0$ and initial covariance $\mathbf{P}_0$ of this state are chosen, and then the posterior state and covariance at step 0 as follows:

$$\begin{aligned} \mathbf{x}_0^+ &= E(\mathbf{x}_0) \\ \mathbf{P}_0^+ &= E[(\mathbf{x}_0 - \mathbf{x}_0^+)(\mathbf{x}_0 - \mathbf{x}_0^+)^T] \end{aligned} \tag{11.9}$$

In the prediction phase, the LKF calculates the prior state estimate, $\mathbf{x}_k^-$ and the prior error covariance, $\mathbf{P}_k^-$ as follows:

$$\begin{aligned} \mathbf{x}_k^- &= \mathbf{A}\mathbf{x}_{k-1}^+ + \mathbf{B}\mathbf{u}_{k-1} \\ \mathbf{P}_k^- &= \mathbf{A}\mathbf{P}_{k-1}^+\mathbf{A}^T + \mathbf{Q} \end{aligned}$$

The update phase estimates the current state based on the prior estimate and the observed measurement:

$$\begin{aligned} \mathbf{G}_k &= \mathbf{P}_k^-\mathbf{C}_k^T(\mathbf{C}_k\mathbf{P}_k^-\mathbf{C}_k^T + \mathbf{R})^{-1} \\ \mathbf{x}_k^+ &= \mathbf{x}_k^- + \mathbf{G}_k(\mathbf{z}_k - \mathbf{C}_k\mathbf{x}_k^-) \\ \mathbf{P}_k^+ &= (\mathbf{I} - \mathbf{G}_k\mathbf{C}_k)\mathbf{P}_k^- \end{aligned}$$

where $\mathbf{x}_k^+$ is the posterior state estimate, $\mathbf{P}_k^+$ is the posterior error covariance, and $\mathbf{G}_k$ is the Kalman gain calculated at each step, which minimizes the trace of the error covariance matrix (Durrant-Whyte, 2001).

The LKF was extended to Extended KF and Unscented KF which are presented in the next sections.

Extended Kalman filter

In reality, the horizontal motion of the VP is nonlinear, because it is derived from the dynamic model of the quadcopter, which is highly nonlinear. Consequently, the LKF may not be the most appropriate estimation method. One solution for nonlinear systems is the EKF.

The EKF linearizes the nonlinear system by taking the first order Taylor approximation around the current state. To this end the Jacobians of the model and sensor system functions are calculated at each step of the algorithm, as shown below:

$$\mathbf{F}_k \triangleq \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x},\, \mathbf{u})\Big|_{\mathbf{x}=\mathbf{x}_k^-\ \mathbf{u}=\mathbf{u}_k} \qquad \mathbf{H}_k \triangleq \frac{\partial h}{\partial \mathbf{x}}(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k^-}$$

The initial state and covariance are set as in (11.9), and the prediction phase calculations become:

$$\begin{aligned} \mathbf{x}_k^- &= f(\mathbf{x}_{k-1}^+, \mathbf{u}_{k-1}) \\ \mathbf{P}_k^- &= \mathbf{F}_k \mathbf{P}_{k-1}^+ \mathbf{F_k}^T + \mathbf{Q} \end{aligned} \tag{11.10}$$

The update phase of the EKF is given as:

$$\begin{aligned} \mathbf{G}_k &= \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \\ \mathbf{x}_k^+ &= \mathbf{x}_k^- + \mathbf{G}_k (\mathbf{z}_k - h(\mathbf{x}_k^-)) \\ \mathbf{P}_k^+ &= (\mathbf{I} - \mathbf{G}_k \mathbf{H}_k) \mathbf{P}_k^- \end{aligned}$$

Unscented Kalman filter

The first order approximation of the EKF derivatives can produce significant errors if f and h are highly nonlinear, thus in the Unscented Kalman Filter (UKF) the linearization around the current state was replaced by a new procedure. The UKF is based on two fundamental principles. First, the nonlinear transformation of a state is easier to approximate than the transformation of a probability density function (pdf). Second, it is possible to generate a set of states whose *pdf* approximation is close to the true state pdf. Using these ideas, a set of states called *sigma points* is generated with the mean equal with $\bar{\mathbf{x}}$ and covariance $\mathbf{P}$ (the covariance of $\mathbf{x}$), and then these states are passed through the nonlinear system function f.

Formally, the system and sensor model are the general ones in (11.3), and the initial state and covariance are chosen as in (11.9). In the prediction phase, if $\mathbf{x}$ has n state variables, then $2n$ sigma point are chosen:

$$\begin{aligned} \hat{\mathbf{x}}^{(i)} &= \hat{\mathbf{x}}_{k-1}^+ + \tilde{\mathbf{x}}^{(i)}, & i &= 1,...,2n \\ \tilde{\mathbf{x}}^{(i)} &= \left(\sqrt{n\mathbf{P}_{k-1}^+}\right)_i^T, & i &= 1,...,n \\ \tilde{\mathbf{x}}^{(n+i)} &= -\left(\sqrt{n\mathbf{P}_{k-1}^+}\right)_i^T, & i &= 1,...,n \end{aligned}$$

where $\hat{\mathbf{x}}^{(i)}$ is the i^{th} estimated sigma point, $\tilde{\mathbf{x}}^{(i)}$ is the error between the true and estimated state, and $\sqrt{\mathbf{P}}$ denotes matrix $\mathbf{Z}$ such that $\mathbf{P} = \mathbf{Z} \cdot \mathbf{Z}$.

The sigma points are transformed into the next state with the nonlinear system transfer function f:

$$\hat{\mathbf{x}}_k^{(i)} = f\left(\hat{\mathbf{x}}_{k-1}^{(i)}, \mathbf{u}_k\right)$$

The prior state estimate is calculated with the approximated mean of the sigma points:

$$\hat{\mathbf{x}}_k^- = \frac{1}{2n} \sum_{i=1}^{2n} \hat{\mathbf{x}}_k^{(i)}$$

The prior error covariance is calculated with the process noise $\mathbf{Q}$ included:

$$\mathbf{P}_k^- = \frac{1}{2n} \sum_{i=1}^{2n} (\hat{\mathbf{x}}_k^{(i)} - \hat{\mathbf{x}}_k^-)(\hat{\mathbf{x}}_k^{(i)} - \hat{\mathbf{x}}_k^-)^T + \mathbf{Q}_{k-1}$$

The measurement update is similar to the time update phase. A new set of sigma points can be generated and propagated, they are denoted with $\mathbf{x}_k^{(i)}$ and $\mathbf{P}_k^{(i)}$. This step can be neglected by using the previously calculated sigma points to reduce the computation time, but sacrificing some performance. The predicted measurements, $\hat{\mathbf{z}}_k^{(i)}$ are calculated with the nonlinear measurement transfer function h: $\hat{\mathbf{z}}_k^{(i)} = h\left(\hat{\mathbf{x}}_k^{(i)}\right)$

In the next step, the covariance matrix of the measurement and the cross covariance between the state and measurement are calculated:

$$\begin{aligned}
\mathbf{P}_{\mathbf{zz}} &= \frac{1}{2n}\sum_{i=1}^{2n}\left(\hat{\mathbf{z}}_k^{(i)} - \hat{\mathbf{z}}_k\right)\left(\hat{\mathbf{z}}_k^{(i)} - \hat{\mathbf{z}}_k\right)^T + \mathbf{R}_k \\
\mathbf{P}_{\mathbf{xz}} &= \frac{1}{2n}\sum_{i=1}^{2n}\left(\hat{\mathbf{x}}_k^{(i)} - \hat{\mathbf{x}}_k^{-}\right)\left(\hat{\mathbf{z}}_k^{(i)} - \hat{\mathbf{z}}_k\right)^T
\end{aligned}$$

The final step is similar to the LKF, where the state is calculated with the Kalman gain:

$$\begin{aligned}
\mathbf{G}_k &= \mathbf{P}_{\mathbf{xz}}\mathbf{P}_{\mathbf{zz}}^{-1} \\
\hat{\mathbf{x}}_k^{+} &= \hat{\mathbf{x}}_k^{-} + \mathbf{G}_k\left(\mathbf{z}_k - \hat{\mathbf{z}}_k\right) \\
\mathbf{P}_k^{+} &= \mathbf{P}_k^{-} - \mathbf{G}_k\mathbf{P}_{\mathbf{zz}}\mathbf{K}_k^T
\end{aligned}$$

At the next step new sigma points are generated based on the posterior estimation, and the above procedure is run again.

Particle filter

The Particle filter (PF) is another probability-based (Gordon et al, 1993; Ristic et al, 2004) estimator as ones above. The method is based on fundamental principles similar to those of the UKF. Basically, PF propagates points trough a nonlinear transformation, rather than propagating a pdf. Initially, N random state vectors are generated based on the starting pdf $p(\mathbf{x}_0)$. The generated state vectors are denoted with $\mathbf{x}_{0,i}^{+}$ and are called particles. One drawback of the PF is the high computational effort.

At each step, the prior particles $\mathbf{x}_{k,i}$ are computed with (11.3) the nonlinear system function f and the process noise, generated randomly with the known pdf. A relative likelihood q_i is computed by evaluating $p(\mathbf{z}_k|\mathbf{x}_{k,i}^{-})$. We introduce the notation $\eta = \mathbf{z} - h(\mathbf{x}_{k,i}^{-})$ as the error of the measurement and prediction. For Gaussian noise the relative likelihood is proportional to:

$$q_i \sim \frac{1}{(2\pi)^{m/2}|R|^{1/2}}\exp\left(-\frac{\eta^T R_i^{-1}}{2}\right)$$

where m is the dimension of the measurement. The relative likelihood is scaled: $q_i = \frac{q_i}{\sum_{j=1}^{N} q_j}$. After generating a set of prior particles $\mathbf{x}_{k,i}^{+}$, the re-sampling step is applied when the effective sample size $N_{eff} = (\sum_{i=1}^{N} w_{k,i}^2)^{-1}$ drops bellow a threshold, where w_k^i is the weight of the i^{th} particle for $\mathbf{x}_k$. For re-sampling, a random number, r is generated uniformly distributed on $[0,1]$. Then, the cumulated sum of q_i is computed until it is less than r and the new particles are $\mathbf{x}_{k,i}^{+} = \mathbf{x}_{k,j}^{-}$ with probability q_j $(i, j = 1,...,N)$. The posterior set of particles $\mathbf{x}_{k,i}^{+}$ is distributed according to the pdf $p(\mathbf{x}_k|\mathbf{z}_k)$. Finally, the mean state is approximated as the average of the posterior set of particles:

$$E\left(\mathbf{x}_k|\mathbf{z}_k\right) \approx \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_{k,i}^{+}$$

11.5 Approach

In this section, we describe out application, starting form the methodology presented in Section 11.4. First, we present the software architecture in Section 11.5.1 and the AR.Drone initialization in Section 11.5.2. Next, we discuss feature detection in Section 11.5.3 and object tracking in Section 11.5.4. Finally, we present the implemented control algorithm in Section 11.5.5.

The corridor following problem was described in (Bills et al, 2011; Gomez-Balderas et al, 2013). A quadcopter should be able to fly through a corridor-like environment by using in this problem only visual perception and the IMU, without using any additional tags or changes in the environment. We extended the solution (Lange et al, 2012) with different filtering algorithms and a simple control strategy, in order to make the quadcopter autonomous not only in indoor but also in outdoor corridor-like environments.

11.5.1 Software Architecture

The AR.Drone firmware is able to communicate with other devices through a wireless connection, using the User Datagram Protocol (UDP). The drone's software capabilities are not designed to be enriched by users, hence our control application will be located off-board for autonomous flight, recall Section 11.3. For this reason, our system has two hardware components: the quadrotor and a personal computer (PC), so the software architecture of this project, presented in Figure 11.5, is composed of these two main parts.

AR.Drone

The drone's SDK is supplying navigational and video data while accepting and executing translational and angular velocity commands, $[v_x, v_y, v_z, \theta_z]$ (Stephane et al, 2012), where θ_z is the yaw angle.

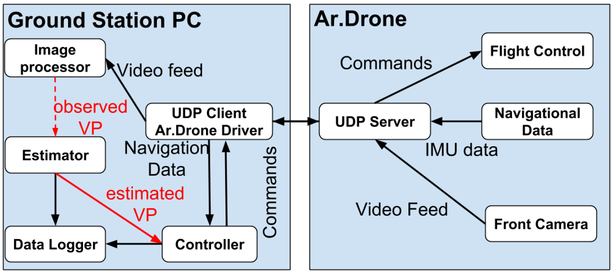


Fig. 11.5: Software architecture, left application on PC (ROS), right AR.Drone.

Ground Station PC

On the PC we are running the ROS framework. In addition, there are functional drivers for the AR.Drone in ROS. We are using the ”ArDrone Autonomy” (Abbas et al, 2013) ROS driver. The autonomous flight controller is written in C++ and runs in ROS. The main nodes in the application are: the *Image processor, Estimator, Data logger,* and finally the *Controller*.

The *Image processor* node detects the VP on the real time video feed and publishes the horizontal position of the VP as an observation in the ROS communication channel. When the node is launched, it initializes the camera calibration parameters. The *Image processor* node is aperiodic, because it waits for a new and calibrated video frame. The route of the video feed starts from the helicopter, where it is packed into a datagram. When the drone driver receives the package, it forwards the image to the image pre-processing node. Then the calibrated image is published in ROS and at the same time the *Image processor* node is notified of the appearance of the new calibrated image.

When a new VP observation is published, the *Estimator* node is announced, in order to estimate the position of the VP based on the observation and/or other navigational data. The estimated horizontal position is published in the common communication channel and it is received by the *Controller* node.

The *Controller* computes velocity commands which are sent back to the drone by the *AR.Drone driver*, hence closing the loop and flying toward the VP. When the controller node is started, it performs sensor calibrations, such as gyroscope offset calibration.

Additionally a *Data Logger* node is created to save all flight data and the control inputs sent to the quadcopter. In the log file we save the packages containing important data, as they appear in the operating systems communication channel. The saved packages are the navigational data, velocity commands, and the calibrated color images from the front camera video feed. This data is used for further analysis and for better tuning.

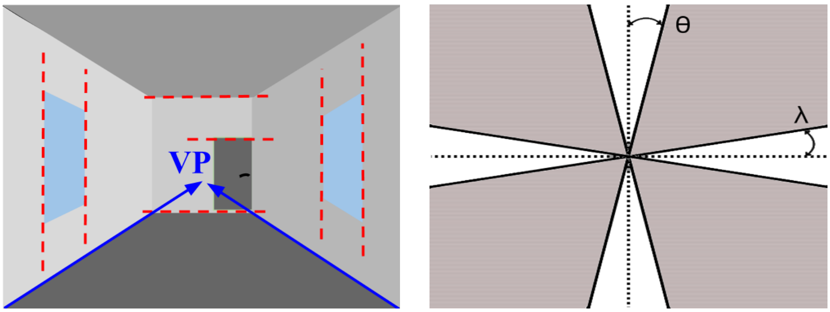
11.5.2 Quadcopter Initialization

The sensors of the quadcopter must be initialized before taking off. More specifically the IMU is calibrated by recalculating the biases. Furthermore the rotation estimates are re-calibrated, hence the UAV must be placed on a flat surface. Finally, the drone stabilizes its X and Y position with the bottom camera by using motion tracking methods. Consequently, it is recommended to use a landing pad with an easy recognizable pattern and wait some seconds before sending any velocity command to the quadcopter.

We used the Monocular Calibration (Remondino and Fraser, 2006) ROS node, to determine the intrinsic and extrinsic parameters of the camera, see Section 11.4.1. The calibration parameters are: $f_x[\text{mm}] = 209.6$, $f_y[\text{mm}] = 210.8$, $c_x[\text{mm}] = 161.2$, $c_y[\text{mm}] = 123.7$, $p_1 = 0.16$, $p_2 = -0.28$, $k_1 \approx 0$, and $k_2 = 0.001$.

11.5.3 Perspective Vision

Earlier, we proposed to use the vanishing point (VP) in order to detect the direction in which the quadcopter should fly. The VP is, by definition, the intersection of parallel lines viewed in perspective, see Figure 11.6. On each frame the edges of the objects are detected as lines.



(a) Detectable lines as noise (b) Line filtering for VP detection

Fig. 11.6: Illustration of VP and VP filtering.

Image processing

Our image processing algorithm finds the VP, which indicates the end of the corridor. The input is the undistorted image and the output is the measurement of the

Fig. 11.7: VP detection in corridors: darker gray rectangle is the VP observation, and the lighter gray rectangle is the estimate of the VP, with the white lines are the detected edges, see Section 11.5.4.

horizontal location of the VP. The image processing algorithm has three steps: edge detection, line detection, and filtering.

For edge detection we used the Canny (Canny, 1986) algorithm, see Section 11.4.1, which is known to have good detection, good localization and minimal response. The detection is considered to be good because the probability is low for both marking false edges and failing to mark real ones. Moreover, the good localization criterion means that the marked edge is close to the center of the real edge. The minimum response property means that an edge should be marked only once, while image noise should not return false detection. The input of the Canny algorithm is a grayscale version of the original image and the output image will contain only the detected edges.

The next phase handles the extraction of long edges by using the PHT algorithm presented in Section 11.4.1. These lines are filtered by their orientation angle, because not all the edges on an image are useful to determine the flight direction, as shown in Figure 11.6a by the red dashed lines. We considered the almost horizontal and vertical lines as noise (not useful for VP detection) in the line detection phase. Hence, all the detected lines having angles in $0 \pm \lambda$, $\pi \pm \lambda$ or $\frac{\pi}{2} \pm \theta$ and $-\frac{\pi}{2} \pm \theta$ are neglected, as presented in Figure 11.6b where the tuning parameters, λ and θ are less or equal with 10°. Furthermore, we divided the image plane into a 23×33 grid and searched for the cell which contains the most intersections obtained from the filtered set of lines. This cell, also shown in Figure 11.7, has high probability to contain the VP. The center of the cell is the observed VP, and will be used as an input for the *Estimator* node.

The PHT threshold refers to the minimum number of detected intersections, see Section 11.4.1 for details. The minimum line length is the threshold for the number of points that can form a line. The maximum line gap is the only parameter that has to be changed for the two different environments: indoor and outdoor. This parameter limits the possible gap between two lines, for which they can be considered as one line.

Ground truth detection

It is essential to have a ground truth, in order to evaluate and compare different approaches and tuning setups. In our case, we must know where exactly is the end of the hall on each frame. For this reason, we decided to place an easily distinguishable object down the hall. The Ground truth detection is not part of our automated flight method. We are using it only in offline mode, to process the logged flight data and evaluate the algorithm.

The tag is composed of tree colored stripes with the same size: the middle one is yellow and the two on the sides are orange. First, the calibrated image goes trough a noise reduction process because of the different brightness values for the same color, while preserving edges. In the next phase we identify the orange and yellow regions with their properties: orientation, centroid, contour, and area. Then we search for the regions of orange, yellow, and orange color "code" as mentioned above. The centroid of the yellow rectangle is considered to be the location of the end of the corridor.

11.5.4 VP Tracking

The observed VP coordinate is processed by the *Estimator* node and the output is the estimated horizontal position of the VP. We need position estimation to filter out faulty VP detection caused by noise on the image and to perform fusion. We are using the three motion models, see Section 11.4.2 to describe the horizontal movement of the VP measurement. The linear Kalman filter is implemented with the linear constant velocity model. The motion of the VP is highly nonlinear, therefore we implemented nonlinear filters: the extended Kalman filter, the unscented Kalman filter, and the particle filter. The nonlinear filters were tested with the linear and the two nonlinear models, specifically the constant velocity model extended with the yaw orientation angle of the quadcopter, and the constant acceleration model.

When VP is not detected, then the estimators are only predicting the position of the VP by skipping the update phase of the algorithm. The covariance matrix $\mathbf{Q}$ is recalculated, see in (11.7), at each step due to the variation of the time duration, δ between two frames.

11.5.5 Control

After having the estimated location of a mobile robot, it should perform its task, in our case the quadcopter should fly along the desired path. The motion of the robot must be regulated with a controller.

We chose a simple strategy,

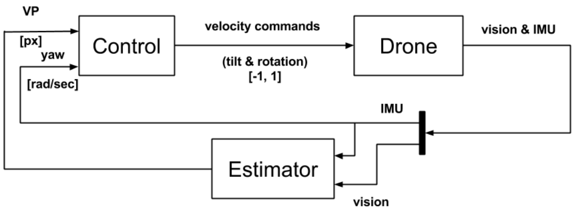


Fig. 11.8: The control loop is composed of the quadcopter as the system , the estimator, and the controller.

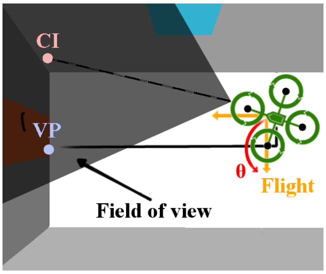


Fig. 11.9: Drone controlled with the switching method. First, the drone corrects the θ angle (red arrow) while hovering. Next, the roll angular velocity is controlled, which produces a lateral translation, in function of the distance between the vanishing point (VP) and the center of the image (CI), while the drone flies forward with a constant velocity. The trajectory (black line) approximately follows the middle of the hallway in this way.

$$u_k = \begin{cases} 0 & \text{, if} \quad e_k < |tr| \\ \text{sign}(e_k) \cdot |u_{max}| & , \quad \text{other case} \end{cases}$$

where, u_k is the control signal at time k, e_k is the error, and u_{max} is the maximum value of the control signal and tr is a threshold value.

We aim to control the horizontal position of the VP on the image, which consequently will generate the desired trajectory of the quadcopter and our control strategy is based on the switching method (Liberzon, 2003). We are switching between 2 controllers, one with higher priority and one with lower priority. The high priority controller regulates the orientation of the drone and it is called **yaw control**. The low priority controller regulates the lateral translational position adn it is called **flight control**.

The block diagram of the control loop is shown on Figure 11.8. The practical functionality of the controller is presented in Figure 11.9, where the quadcopter is moved from the center of the hall and its yaw angle is not aligned with the reference angle.

Yaw control
We defined the initial position of the drone, where the front camera is looking down the hall and the drone is positioned on the middle of the corridor. In the initialization stage, the rotational angle yaw around the $\mathbf{Z}$ axis, θ is fixed and considered to be the reference signal for the controller. In regular mode, the θ control holds the angle within a small range around this reference by using a maximum gain $u_\theta = 0.02$. This is a prerequirement for the flight control sage. The measurement of θ is obtained form the navigational data. The yaw angle can suffer from drift, but it maintained fine in our experiments

Flight control
The flight controller generates the translational velocity commands, which are sent to the drone. The lateral movement is controlled with a maximum gain $u_{pitch} = 0.015$, while the forward movement is constant, $u_{roll} = 0.065$. Moreover, the controller is responsible with landing the UAV in case the image processing algorithm dose not find the end of the corridor. The drone is landed if on 20 consecutive frames no VP can be observed. This situation can happen in case the drone is at the end of the hall (facing the wall, so 0 lines detected), but this is also a safety measurement when the drone looses sight of the end of the hallway.

11.6 Experiments and Results

In this section we present the test conditions and algorithm setups for the performed experiments. We analyzed the Canny algorithm with the goal of noise reduction on outdoor images. We examined the linear and nonlinear state estimators with the three model approximations of the VP motion on the image and performed control experiments in real indoor and outdoor corridor-like environments.

The experiments were conducted in both indoor corridors and outdoor hallway-like environments. We ran multiple tests on corridors with different lengths and widths. The conditions for these experiments are:

- the corridor is straight with length $\in [22\ 30]$ meters and width $\in [1.8\ 2.7]$ meters;
- it can have connections with other corridors;
- there are no obstacles;
- the quadcopter is initially placed on the middle of the corridor facing the end of the hall;
- the yaw angle is initially aligned with the direction of the hallway.
- the quadcopter has the outdoor hull

The ground truth was obtained with the object detector application, see Section 11.5.3 for more details.

The image processing parameters were tuned with real scenario experiments, while the VP position estimation parameters for each filter were tuned offline in Matlab by using the recorded sensor data, the observed VP position, and the remarked ground truth. The control parameters were tuned based on both real and simulated environments.

We tuned the edge and line detection algorithms experimentally for both indoor and outdoor corridor-like scenes and the results are shown in Table 11.1.

Table 11.1: Image processing parameters for indoor and outdoor hallway like environments.

	Indoor	Outdoor
Canny mask size	3	
Canny minimum threshold	50	
Canny maximum threshold	100	
PHT threshold	65	
PHT minimum line length	15	
PHT maximum line gap	45	85

11.6.1 VP Motion Model Results

We analyzed the three motion models, constant velocity, constant velocity extended with the yaw orientation angle and constant acceleration, in combination with the LKF with the same standard deviation of the process $\sigma_w = 8$[px] and for the observation $\sigma_v = 70$[px]. The nonlinear model is linearized around the equilibrium point (in 0) in order to use the LKF, as presented in Section 11.4.2. These experiments were performed offline in Matlab, using the logged flight data form real experiments. The results of these experiments are shown in Figure 11.10.

We observed that the constant velocity model has the best performance for this linear estimator. This is likely because, a linear filter should be used with a linear model to reach the expected outcome. The nonlinear model does not perform very badly, but increases the computational time without giving better estimation error.

11.6.2 Nonlinear Estimator Results

In this section we give the performance evaluation for the three nonlinear filters, specifically for EKF, UKF, and PF.

We were interested in the performance of different nonlinear estimators, because the dynamic motion of the VP is highly nonlinear, despite the use of our simplified models. In these experiments we were using the constant velocity model extended with the yaw angle. The tests were performed offline in Matlab on a standard PC.

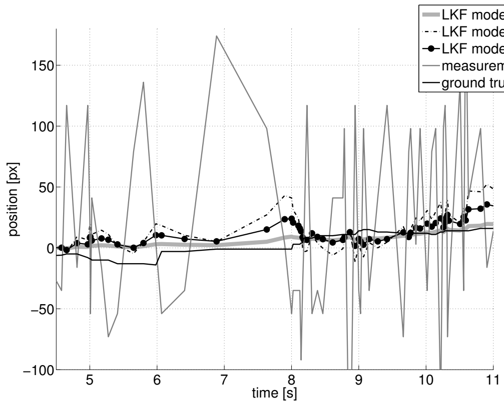


Fig. 11.10: Comparison of three motion models used in LKF. The horizontal axis is the time and the vertical axis is the horizontal position of the VP. The black line is the ground truth, the dashed is the observed VP. Model 1 is the constant velocity model (11.4), model 2 is the constant acceleration model (11.6), and model 3 is the constant velocity model extended with the yaw angle of the quadcopter (11.5).

The estimator's input is real measurement data, while the output is the estimated horizontal position of the VP.

The PF algorithm has an extra parameter compared to the nonlinear Kalman estimators ($\sigma_w = 8$[px] and $\sigma_v = 70$[px]), namely the number of particles. We tuned this parameter to be 51 because the mean estimation error stabilizes around 8[px] at this value while keeping the computational time less than 8 ms.

The outcomes are shown in Table 11.2. We present graphically in Figure 11.11 the behavior of all the three nonlinear filters for the same test flight in an indoor hallway. The EKF and UKF methods gave similar results, but the time performance of the UKF algorithm is slightly better than the EKF. The PF method estimation precision is the weakest compared to the other two methods and the algorithm also needs more computational power and time.

11.6.3 Indoor and Outdoor Results

In this section we compare the VP detection and estimation performance between the indoor and outdoor parameter setup in the detection phase of the algorithm. The experiments were carried out in an outdoor hallway-like environment where we compared both indoor and outdoor parameter settings. We are not investigating the performance of disturbance rejection due to different illumination and wind speed. We only aim to reduce the noise in the VP position caused by the increased texture complexity outdoor compared to an indoor scene.

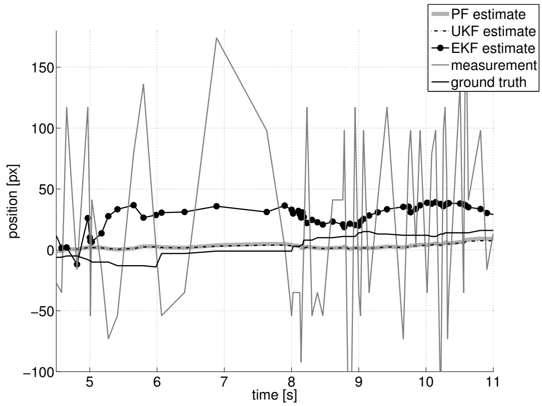


Fig. 11.11: Nonlinear filters comparison of the estimated horizontal position of the VP with nonlinear model (11.5). The black line is the ground truth, the dashed line is the observation, the line with dots is the EKF estimation, the gray line is the PF estimation, and the dotted line is the UKF estimate.

Table 11.2: VP estimation performance with EKF, UKF, and PF, in case of nonlinear model (11.5) and outdoor flight with outdoor detection setup. Obs is the observation, t_R time period for an estimation, std is standard variation.

	Obs	EKF	UKF	PF
std [px]	82.12	8.49	8.49	8.43
mean [px]	-16.83	2.65	2.66	3.10
mode [px]	38	-16.41	-16.41	-15.94
median [px]	-10	6.87	6.88	7.31
t_R [ms]	-	0.61	0.51	7.55

The increase of the precision in estimation from the LKF to the UKF can be found in Table 11.3, where the linear simplified model (11.4) was used, and the values represent the error calculated from the difference between the ground truth and the estimation, t_R is the elapsed time while one estimation is done, std is standard variation. The results are showing an increase of estimation performance for the UKF, but also reveals longer execution time weakness compared to the LKF.

Table 11.3: VP estimation performance errors with LKF and UKF, in case of linear model (11.4) and outdoor flight with in- and out-door detection parameters.

	Indoor setup		Outdoor setup	
	LKF	UKF	LKF	UKF
std [px]	22.34	6.29	19.40	8.52
mean [px]	-34.73	-2.26	-11.22	2.65
mode [px]	-76.52	-17.97	-55.49	-16.43
median [px]	-30.17	-1.31	18.83	6.79
t_R [ms]	0.19	0.48	0.19	0.49

11.6.4 Control Results

The maximum control values and thresholds were chosen experimentally, see Table 11.4, keeping in mind that halls usually are quite narrow, so smooth lateral and rotation movements are preferred.

Parameters	Values
Forward velocity	0.065
Lateral velocity	0.015
Yaw rotational velocity	0.1
VP position threshold	5 [px]
Yaw angle threshold	2 [°]

Table 11.4: Control and threshold parameters.

The controller node was tested in both real and simulated environments. The trajectory of the drone in the simulation was perfectly identical with the center line of the hallway, because of the lack of noise in the VP detection. Consequently, we performed test flights on real corridors, the results of which can be seen in the online video[4]. The flight tests were successful, indoor in Figure 11.12 and also outdoor see Figure 11.13, the drone stabilizes the yaw angle and flies through the hall. We observed that in narrow corridors the drone has difficulties in stabilization because of the changes in the aerodynamics.

We can observe in Figure 11.13 that the initial yaw angle is displaced in regard of the reference value and while this is compensated by the **yaw control** node, the VP shifts further to the left on the image. This drift of the VP is corrected between the 12th and 20th second of the experiment.

[4] video link: https://www.youtube.com/watch?v=S7VQWP7O91k

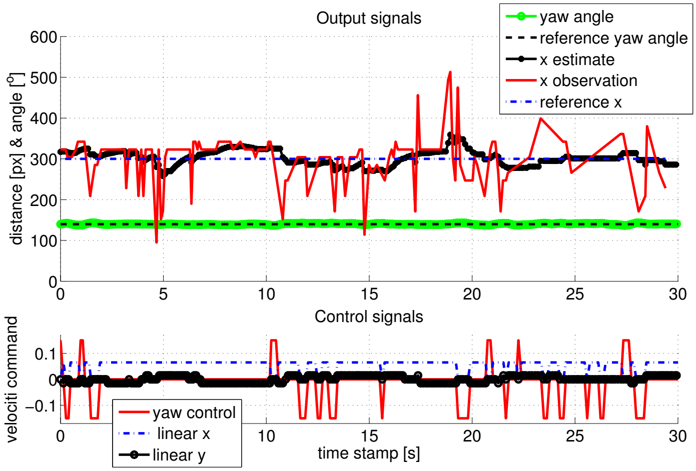


Fig. 11.12: Indoor flight results, the upper graphic shows the output signals: yaw angle and its reference angle, the estimated and observed observed horizontal pixel coordinate of the VP and its reference value. On the bottom are the control signals: yaw angle, linear x or ahead and linear y or lateral velocity commands.

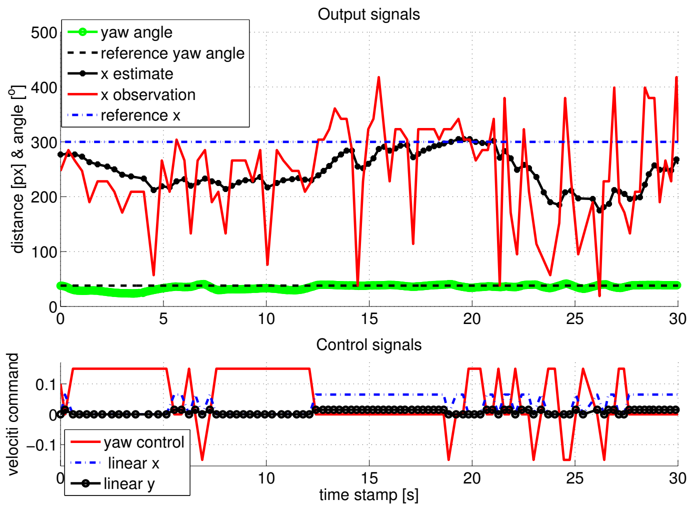


Fig. 11.13: Outdoor flight results, similarly to the indoor results in Figure 11.12.

11.7 Summary & Perspectives

In this chapter we showed an approach for the autonomous flight control of a quadcopter in hallway-like environments. We used low-level image processing enhanced with *probabilistic filters* which supports data fusion, while using a low-cost quadcopter.

The approach presented for hallway-like environments can be generalized for other similar conditions. The experimental results already show that the same image processing algorithm can be used for both indoor and outdoor environment by slightly changing some of the tuning parameters.

Therefore, based on the presented approach in this chapter, we have already applied our idea to another application, railway track following, which uses similarly the vanishing point to follow the rail tracks (Pall et al, 2014). We already achieved good results in simulation and our next aim is to perform experiment on real rail tracks. We are going to extend also the simple yaw angle model with the quadcopter dynamic model.

References

Abbas S, Zhao S, Mani M, Richard V (2013) AutonomyLab, Ar.Drone drivere in ROS. URL `github.com/AutonomyLab/ardrone_autonomy`

Ali M (2008) Real time detection of lane markers in urban streets. In: Proceedings on Intelligent Vehicles Symposium, IEEE, Eindhoven, the Netherlands, pp 7–12

Beard C, Chen ZQ, Kumar V, Lee Y, Leon-Salas WD, Rao P (2013) SAVEUS: Saving Victims in Earthquakes through Unified Systems. IJCNDS 10(4):402–420

Benini A, Mancini A, Longhi S (2013) An IMU/UWB/Vision-based Extended Kalman Filter for Mini-UAV Localization in Indoor Environment using 802.15.4a Wireless Sensor Network. Journal of Intelligent & Robotic Systems 70(4):461–476

Bills C, Chen J, Saxena A (2011) Autonomous MAV flight in indoor environments using single image perspective clues. In: IEEE International Conference on Robotics and Automation, IEEE, pp 5776–5783

Bonin-Font F, Ortiz A, Oliver G (2008) Visual Navigation for Mobile Robots: A Survey. Journal of Intelligent Robotics Systems 53(3):263–296

Bresciani T (2008) Modeling, Identification and Control of a Quadrotor Helicopter. Master's thesis, Lund University, Sweden

Canny J (1986) A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence (6):679–698

Durrant-Whyte HF (2001) Introduction to estimation and the Kalman Filter. Australian Centre for Field Robotics pp 25–29

Gomez-Balderas JE, Flores G, Carrillo LG, Lozano R (2013) Tracking a ground moving target with a quadrotor using switching control. Journal of Intelligent & Robotic Systems 70(1-4):65–78

Gordon N, Slamond D, Smith A (1993) Novel approach to nonlinear/non-Gaussian Bayesian state estimation. IEEE Proceedings on Radar and Signal Processing pp 107–113

Hua MD, Hamel T, Morin P, Samson C (2013) Introduction to feedback control of underactuated VTOL vehicles: A review of basic control design ideas and principles. Control Systems, IEEE 33(1):61–75

Kiryati N, Eldar Y, Bruckstein AM (1991) A probabilistic Hough transform. Pattern recognition 24(4):303–316

Krajník T, Vonásek V, Fišer D, Faigl J (2011) AR-drone as a platform for robotic research and education. In: Research and Education in Robotics-EUROBOT, Springer, pp 172–186

Lange S, Sünderhauf N, Neubert P, Drews S, Protzel P (2012) Autonomous corridor flight of a UAV using a low-cost and light-weight RGB-D camera. In: Advances in Autonomous Mini Robots, Springer, pp 183–192

Liberzon D (2003) Switching in systems and control. Springer

Liou SP, Jain RC (1987) Road following using vanishing points. Computer Vision, Graphics, and Image Processing 39(1):116–130

Majdik A, Albers-Schoenberg Y, Scaramuzza D (2013) MAV urban localization from Google street view data. In: International Conference on Intelligent Robots and Systems, IEEE, pp 3979–3986

Odido MD, Madara D (2013) Emerging Technologies: Use of Unmanned Aerial Systems in the Realisation of Vision 2030 Goals in the Counties. International Journal of Applied 3(8)

Pall E, Mathe K, Tamas L, Busoniu L (2014) Railway track following with the AR. Drone using vanishing point detection. In: Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on, IEEE, pp 1–6

Pavlik JV (2014) Transformation: Examining the Implications of Emerging Technology for Journalism, Media and Society. Athens Journal of Mass Media and Communications

Peterfreund N (1999) Robust tracking of position and velocity with Kalman snakes. IEEE Transactions on Pattern Analysis and Machine Intelligence 21(6):564–569

Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng AY (2009) ROS: an open-source Robot Operating System. In: ICRA workshop on open source software, vol 3, p 5

Remondino F, Fraser C (2006) Digital camera calibration methods: considerations and comparisons. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 36(5):266–272

Ristic B, Arulampalm S, Gordon NJ (2004) Beyond the Kalman filter: Particle filters for tracking applications. Artech House Publishers

Rubinsztejn Y (2011) Automatic Detection of Objects of Interest from Rail Track Images. Master's thesis, University of Manchester

Schulz D, Burgard W, Fox D, Cremers AB (2001) Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. In: IEEE International Conference on Robotics and Automation, IEEE, vol 2, pp 1665–1670

Shen S, Mulgaonkar Y, Michael N, Kumar V (2013) Vision-based state estimation for autonomous rotorcraft MAVs in complex environments. In: IEEE International Conference on Robotics and Automation, IEEE, pp 1758–1764

Stephane P, Nicolas B, Pierre E, Frederic DH (2012) AR.Drone Developer Guide

Stephens RS (1991) Probabilistic approach to the Hough transform. Image and vision computing 9(1):66–71

Tomic T, Schmid K, Lutz P, Domel A, Kassecker M, Mair E, Grixa I, Ruess F, Suppa M, Burschka D (2012) Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue. Robotics Automation Magazine, IEEE 19(3):46–56

Zhang Z (2000) A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(11):1330–1334